Graphics programming in Icon

This quick reference guide to Icon graphics is taken from Graphics facilities for the Icon Programming Language, Version 9.1, by Clinton L. Jeffery, document number IPD268, date November 1, 1995. This document is available in compressed PostScript form at:

Note: You must include the statement
link graphics
in your Icon source program.

Graphics functions and procedures

These conventions are used to identify arguments:

- \[ W \] A window. Any such argument can be omitted, in which case the subject window &window is used.
- \( a \) Any type.
- \( x, y \) Coordinate location.
- \( w, h \) Width and height.
- \( \theta, \alpha \) Angles, real numbers measured in radians.
- \( k \) Color specification, string or integer.

Calling sequences

Many of the calls are used to create and modify canvases and graphics contexts (GCs):

- **Canvas** attributes are properties associated with the actual rectangle on the screen where the graphics appear, for example, its size and title.

- **Graphics context** attributes are properties that affect how you display things in a canvas, for example, foreground colors, line widths, and text fonts.

Either a canvas or a GC can be passed as a \[ W \] (window) argument.

Attribute functions

These functions can be used to set or query the attributes of either a canvas or graphics context.

\[ WAttrib(W, s_1, s_2, \ldots) \]

Each \( s_i \) can be either a string of the form "\( a=v \)"", which means set attribute \( a \) to value \( v \), or just "\( a \)". First all attributes \( a_i \) are set to the corresponding values \( v_i \) if given; then the function generates the values of all the \( a_i \).

Example: \[ WAttrib("label=Picture Window") \] would set and return the title of the current window, and \[ WAttrib("label") \] would just return that title.
**Canvas attributes**

The superscript $^R$ indicates attributes that can only be read (queried); superscript $^W$ indicates those that can only be written (set). Attributes without those notations can be set or queried.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>canvas</td>
<td>Window visibility: &quot;hidden&quot;, &quot;iconic&quot;, &quot;normal&quot;, &quot;maximal&quot;</td>
</tr>
<tr>
<td>col</td>
<td>Current text cursor column number</td>
</tr>
<tr>
<td>columns</td>
<td>Width of the canvas in characters</td>
</tr>
<tr>
<td>cursor</td>
<td>Cursor visibility, &quot;on&quot; or &quot;off&quot;</td>
</tr>
<tr>
<td>depth$^R$</td>
<td>Bits per pixel in this graphics system</td>
</tr>
<tr>
<td>display</td>
<td>The name of the display monitor</td>
</tr>
<tr>
<td>displayheight$^R$</td>
<td>Height of the entire screen in pixels</td>
</tr>
<tr>
<td>displaywidth$^R$</td>
<td>Width of the entire screen in pixels</td>
</tr>
<tr>
<td>echo</td>
<td>Controls echoing of keyboard input, &quot;on&quot; or &quot;off&quot;</td>
</tr>
<tr>
<td>height</td>
<td>Height of the canvas in pixels</td>
</tr>
<tr>
<td>iconimage</td>
<td>The name of an image file to be used as this window’s icon when it is iconified</td>
</tr>
<tr>
<td>iconlabel</td>
<td>The label to appear on the icon when this window is iconified</td>
</tr>
<tr>
<td>iconpos$^W$</td>
<td>Position of the iconified window as a string &quot;$x,y$&quot;</td>
</tr>
<tr>
<td>image$^W$</td>
<td>The name of an image file to be used as the initial window contents when opened</td>
</tr>
<tr>
<td>label</td>
<td>The window’s title</td>
</tr>
<tr>
<td>lines</td>
<td>Height of the canvas in characters</td>
</tr>
<tr>
<td>pointer</td>
<td>Describes the shape of the mouse pointer; names are system-dependent.</td>
</tr>
<tr>
<td>pointercol</td>
<td>Mouse location as a text column</td>
</tr>
<tr>
<td>pointerrow</td>
<td>Mouse location as a text row</td>
</tr>
<tr>
<td>pointerx</td>
<td>Mouse $x$ position in pixels</td>
</tr>
<tr>
<td>pointery</td>
<td>Mouse $y$ position in pixels</td>
</tr>
<tr>
<td>pos</td>
<td>Window position on screen as a comma-separated string &quot;$x,y$&quot;</td>
</tr>
<tr>
<td>posx</td>
<td>Window $x$ position</td>
</tr>
<tr>
<td>posy</td>
<td>Window $y$ position</td>
</tr>
<tr>
<td>resize</td>
<td>Controls whether the user is allowed to resize the window; &quot;on&quot; or &quot;off&quot; (the default, meaning no resizing)</td>
</tr>
<tr>
<td>row</td>
<td>Text cursor position in rows</td>
</tr>
<tr>
<td>size</td>
<td>Window size in pixels as a string &quot;$x,y$&quot;</td>
</tr>
<tr>
<td>width</td>
<td>Window width in pixels</td>
</tr>
<tr>
<td>x</td>
<td>Mouse $x$ position in pixels at the last event</td>
</tr>
<tr>
<td>y</td>
<td>Mouse $y$ position in pixels at the last event</td>
</tr>
</tbody>
</table>

**Graphics context (GC) attributes**

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ascent$^R$</td>
<td>Height in pixels of the current text font, above the baseline</td>
</tr>
<tr>
<td>bg</td>
<td>Background color; default white</td>
</tr>
<tr>
<td>cliph</td>
<td>Height of clipping region</td>
</tr>
<tr>
<td>clipw</td>
<td>Width of clipping region</td>
</tr>
</tbody>
</table>
clipx  Left side of clipping region
clippy  Right side of clipping region
descent  Depth in pixels of the current font, below baseline
drawop  How newly drawn pixels are combined with existing pixels, "copy" or "reverse" (exclusive-or)
dx  Horizontal translation offset; with dx, specifies the effective origin within this GC
dy  Vertical translation offset
fg  Foreground color
fheight  Overall height in pixels of the current font
fillstyle  Drawing function for area filling, "solid", "textured", or "masked"
font  Current text font
fwidth  Width in pixels of the widest character in the current font
leading  The baseline-to-baseline vertical spacing for multiline text messages
linestyle  Controls line style, "solid", "dashed", or "striped"
linewidth  The line width, default 1 (in pixels)
pattern  The fill pattern used when fillstyle is "textured" or "masked". Values include "black" (the default), "very-dark", "darkgray", "gray", "lightgray", "very-light", "white", "vertical", "diagonal", "horizontal", "grid", "trellis", "checkers", "grains", "scales", or "waves"
reverse  If "on", drawing uses the background color and erasure uses the foreground color; default is "off"

**Window (canvas) functions**

WOpen(W, s1, s2, ...)
Creates a new window. If any si are given, they are used to set that window’s attributes as in WAttrib(). Returns a window object.

GotoRC(W, r, c)
Set the text cursor position to row r and column c, where (1, 1) is in the upper left corner.

GotoXY(W, x, y)
Set the text cursor position to coordinate (x, y).

Lower(W)
Put the window behind all other windows.

Pixel(W, x, y, w, h)
Generates the colors of the pixels in the given rectangle, left to right, then top to bottom.

Raise(W)
Move the window in front of all other windows.

Uncouple(W)
Frees graphics context W. If there are no other GCs to this canvas, the window is closed.

WClose(W)
Close the canvas given by W.

WDefault(W, s1, s2)
Gets the default value of option s2 for program s1 in the graphics system.

WDelay(W, i)
Flushes any pending output to window W, then waits for i milliseconds and returns.

WDone(W)
 Waits for a "q" or "Q" to be entered, and then terminates the program.

WFlush(W)
Forces any output to window W.
WQuit(W)  If the event queue contains the entry of a "q" or "Q" to be entered, removes events up to and including that event and then returns to the caller. Fails if there is no such event.

WSync(W)  Synchronizes the display with the graphics server and then returns.

**Graphics context functions**

Bg(W,k1)  Returns the background color. If k1 is given, sets the background to that color.

Clip(W,x,y,w,h)  Set the clipping rectangle. With no arguments, clip to the edge of the canvas.

Clone(W,s1,...)  Returns a window that shares a canvas with W but has different attributes; each S_i has the form "attr=value".

Color(W,i1,k1,...)  If k1 is given, set mutable color i1 to that color, else return the color of mutable color i1. You may supply any number of i and k pairs.

ColorValue(W,k)  Returns a string "r,g,b" with the red, green, and blue values as integers in [0,65535].

Couple(W1,W2)  Return a new window with the canvas from W1 and the graphics context of W2.

Fg(W, k)  If k is given, sets the foreground color in W. In any case, returns the foreground color.

Font(W,s)  If s is given, the font is set to that name. Returns the current font in W. Fonts are specified as a string of names, styles, and sizes in pixels. For example, "Times,bold,italic,18". Exact font availability is system dependent, but these font families are always available: two monospaced fonts named mono (sans-serif) and typewriter (serifed), and two proportional fonts named sans (sans-serif) and serif (serifed).

FreeColor(W,k,...)  Dellocate the given color(s).

NewColor(W,k)  Allocates color k and returns a small negative integer that can be passed as a color. Fails if no entries remain in the colormap.

Pattern(W,s)  Sets the fill pattern for fill styles "masked" or "textured".

**Drawing functions**

The handling of images and patterns is too complex to be discussed here; refer to document IPD268 for full particulars.

CenterString(W,x,y,s)  Draw string s centered vertically and horizontally at (x,y).

CopyArea(W1,W2,x,y,w,h,xo,yo)  Copy a rectangle from (x,y) to (xo,yo).

DrawArc(W,x,y,w,h,\theta,\alpha)  Draw an elliptical arc inscribed in the given rectangle and subtending the angles from \theta to \theta + \alpha.

DrawCircle(W,x,y,r,\theta,\alpha)  Draw a circular arc centered at (x,y) and subtending the angles from \theta to \theta + \alpha.
DrawCurve($W, x_1, y_1, \ldots$) Draw a smooth curve through all the $(x_i, y_i)$ points. For a closed curve, repeat the starting point at the end of the list.

DrawImage($W, x, y, s$) Draw an image in the rectangle starting at $(x, y)$. Refer to the full specification for the specification of the string $s$. Returns a positive integer if one or more colors could not be allocated, else returns $null$.

DrawLine($W, x_1, y_1, \ldots$) Connect all pairs of points.

DrawPoint($W, x, y, \ldots$) Draw a point at $(x, y)$; any number of pairs may be supplied.

DrawPolygon($W, x_1, y_1, \ldots$) Draw a polygon from $(x_1, y_1)$ through all points given and then back to $(x_1, y_1)$.

DrawRectangle($W, x, y, w, h$) Draw the perimeter of a rectangle.

DrawSegment($W, x_1, y_1, x_2, y_2, \ldots$) Draw line segment from $(x_1, y_1)$ to $(x_2, y_2)$. The last four arguments may be repeated to draw multiple, disconnected line segments.

DrawString($W, x_0, y_0, s, \ldots$) Draw string $s$ with its upper left corner at $(x_0, y_0)$. Any number of strings may be supplied.

EraseArea($W, x, y, w, h$) Set the given rectangle to the background color. Default is the whole window.

FillArc($W, x, y, w, h, \theta, \alpha$) Same as DrawArc(), etc., but fills the area.

FillCircle($W, x, y, r, \theta, \alpha$) Draw a circle with center $(x, y)$ and radius $r$.

FillPolygon($W, x_1, y_1, \ldots$) Draw a polygon from $(x_1, y_1)$ through all points given.

FillRectangle($W, x, y, w, h$) Fill the given rectangle.

LeftString($W, x_0, y_0, s$) Draw string $s$ flush left against $x = x_0$ and centered around $y = y_0$.

ReadImage($W, s_1, x, y, s_2$) Load an image from file $s_1$ and place its upper left corner at $(x, y)$.

RightString($W, x_0, y_0, s$) Draw string $s$ flush right against $x = x_0$ and centered around $y = y_0$.

TextWidth($W, s$) Returns the width of string $s$ in pixels if it were drawn in window $W$.

WRead($W$) Like read(), but returns strings typed in window $W$.

WReads($W, i$) Like reads(), but returns strings typed in window $W$.

WriteImage($W, s, x, y, w, h$) Writes a .gif file named $s$ using the given rectangle of $W$.

WWrite($W, s_1, s_2, \ldots$) Writes the given strings $s_i$ to the window, with the text in the foreground color and the area behind the text in the background color. An implied newline is added at the end.

WWrites($W, s_1, s_2, \ldots$) Same as WWrite(), but without the implicit newline at the end.
Event-handling functions

Refer to document IPD268 for the exact format of events.

**Active()**
Returns a window with a pending event, waits otherwise.

**Alert(W)**
Beeps or flashes the window.

**Event(W)**
Waits for, and returns, the next event. Sets &x, &y, &row, &col, &interval, &control, &shift, and &meta.

**Pending(W)**
Returns the event list for the window—an empty list if there are no events.

Dialog box functions

**ColorDialog(W,L,k,p,a)**
Display a dialog box to be used for color selection. The box has Okay and Cancel buttons. \( L \) is an optional string or list of caption strings. \( k \) is the starting color, defaulting to the foreground color. \( p \) is an optional callback procedure, called as \( p(a,s) \) whenever the color is changed, where \( s \) is the new color setting (as returned by ColorValue). Upon return, the global variable dialog_value will be set to the final color, and the function’s return value will be the name of the button selected.

**Dialog(W,L₁,…,L₅,i)**
Display a dialog box and wait for a button press. Each \( Lᵢ \) can be omitted, given as a single element, or given as a list. \( L₁ \) is a list of zero or more captions. The next three lists describe zero or more text entry fields. For the \( i \)th field, \( L₂[ᵢ] \) is the label, \( L₃[ᵢ] \) is the default value, and \( L₄[ᵢ] \) is the maximum width (default 10). List \( L₅ \) describes the buttons as strings; the default is a single button labeled "Okay". The return value is the name of the button used to exit the dialog, and the global variable dialog_value is set to a list of the values of any text fields.

**Notice(W,s₁,s₂,…,sₙ)**
Opens a dialog box with an “Okay” button and displays all the strings \( sᵢ \) in it. When the user clicks on the “Okay” button, it returns.

**OpenDialog(W,s₁,s₂,i)**
Displays a text dialog box with caption string \( s₁ \) and initial value \( s₂ \) (optional), allowing entry of up to \( i \) characters. The box has “Okay” and “Cancel” buttons, and the function returns the name of the button used, and sets global variable dialog_value to the text entered by the user.

**SaveDialog(W,s₁,s₂,i)**
Similar to OpenDialog(), but has “Yes” and “No” buttons instead of “Okay.”
SelectDialog($W, L_1, L_2, s, L_3, i$)
Displays a dialog box with captions from list $L_1$, zero or more radiobuttons with captions from $L_2$, and buttons with names taken from $L_3$. $s$, if given, selects the default radiobutton name. $i$, if given, specifies the default button number, 0 for no default button. The function returns the name of the button used to exit the dialog, and sets global $dialog_value$ to the name of the selected radiobutton.

TextDialog($W, L_1, L_2, L_3, L_4, L_5, i$)
Displays a dialog box with captions from $L_1$; zero or more text entry fields whose labels are given by elements of $L_2$, whose default values are given by elements of $L_3$, and whose maximum widths are given by elements of $L_4$; and a list of buttons whose names are taken from $L_5$. If $i$ is given, it is the index of the default button (0 for no default). The function returns the name of the exit button, and sets $dialog_value$ to the a list of the values entered into the text fields.

ToggleDialog($W, L_1, L_2, L_3, L_4, i$)
Displays a dialog box with captions from list $L_1$; zero or more checkboxes (“toggle buttons”) whose names are given by elements of $L_2$, whose default values are given by elements of $L_3$ (1 or &null); and buttons whose labels are given by $L_4$. Returns the name of the exit button, and sets $dialog_value$ to a list of toggle button states (1 or &null).

**Graphics keywords**

These keywords are used by the graphics system:

- $\&col$ The column location (as in GotoRC()) of the mouse at the last event.
- $\&control$ If the control key was down during the last event, &null, else fails.
- $\&interval$ The interval in milliseconds between the last two events.
- $\&ldrag$ Use this as a case label for a left-button drag event.
- $\&lpress$ Use this as a case label for a left-button click event.
- $\&lrelease$ Use this as a case label for a left-button release event.
- $\&mdrag$ Use this as a case label for a middle-button drag event.
- $\&meta$ If the meta key was down during the last event, &null, else fails.
- $\&mpress$ Use this as a case label for a middle-button click event.
- $\&mrelease$ Use this as a case label for a middle-button release event.
- $\&rdrag$ Use this as a case label for a right-button drag event.
- $\&rpress$ Use this as a case label for a right-button click event.
- $\&row$ The row location (as in GotoRC()) of the mouse at the last event.
<table>
<thead>
<tr>
<th>Stack name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&amp;rrelease</td>
<td>Use this as a case label for a right-button release event.</td>
</tr>
<tr>
<td>&amp;shift</td>
<td>If the shift key was down during the last event, &amp;null, else fails.</td>
</tr>
<tr>
<td>&amp;window</td>
<td>The current default window.</td>
</tr>
<tr>
<td>&amp;x</td>
<td>The x coordinate of the mouse at the last event.</td>
</tr>
<tr>
<td>&amp;y</td>
<td>The y coordinate of the mouse at the last event.</td>
</tr>
</tbody>
</table>